PŻ

DJANGO 03

KONTYNUUJEMY SZABLONY

```
def spis_pacjentow(request):
    wszyscy = Pacjent.objects.all()
    return render(request, template_name: 'pacjenci.html', context: {'imiona_pacjentów': wszyscy}) #wyś
```

Przypomnienie:

Pobieramy modele z bazy danych, po czym

Przekazujemy do szablonu. W szablonie

Definiujemy co dalej

KONTYNUUJEMY SZABLONY

```
{% for imiona_pacjentów in imiona_pacjentów%}
<p>{{ imiona_pacjentów }}</p>
{% endfor %}
#dodane dynamiczne przekazywanie pojedynczych danych - taka niewielka pętla
</body>
</html>
```

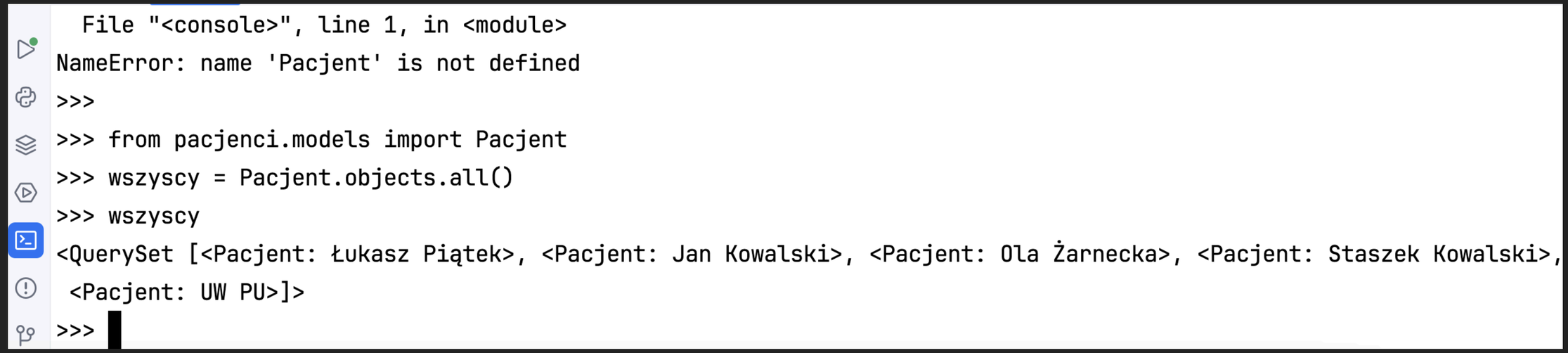
```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0,
 maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <h2>To beda moje filmy</h2>

    {% for film in filmy %}
        <p>{{ film.tu }}</p>
    {% endfor %}

</body>
</html>
```

html › body › p

Skoro w objecy było all
to jest dostęp do wszystkiego.
Po kropce można wybierać co
wyświetlamy

WARTO WSPOMNIEĆ O ORM

- Coś, co ułatwia nam, język, który tłumaczy kod na język bazy danych.



WARTO WYKORZYSTYWAĆ SHELL DO PRACY

- Mowa o konsoli pythona. Jeżeli nasze IDE nie ma tej opcji - to wpisujemy w terminalu python3 manage.py shell i tyle

```
>>> wszyscy
<QuerySet [<Pacjent: Łukasz Piątek>, <Pacjent: Jan Kowalski>, <Pacjent: Ola Żarnecka>, <Pacjent: Staszek Kowalski>,
 <Pacjent: UW PU>]>
>>> wszyscy[1]
<Pacjent: Jan Kowalski>
>>> wszyscy[1].Imię
'Jan'
>>>
```

WARTO WYKORZYSTYWAĆ SHELL DO PRACY

- Możemy przeglądać, ale też i zarządzać obiektami bazy danych. Można szukać.
- A na przykład zamiast poniższego get, dać filter, zrobić np komendę len(wszyscy) - policzyć

```
<QuerySet [<Pacjent: Łukasz Piątek>, <Pacjent: Jan Kowalski>, <Pacjent: Ola Żarnecka>, <Pacjent: S
 <Pacjent: UW PU>]>
>>> wszyscy[1]
<Pacjent: Jan Kowalski>
>>> wszyscy[1].Imię
'Jan'
>>> jeden = Pacjent.object
```

Menedżer, który pozwala na przeglądanie ale też zarządzanie bazą danych

```
>>> jeden = Pacjent.objects.get(Imię='Jan')
>>> jeden
<Pacjent: Jan Kowalski>
>>>
```

WARTO WYKORZYSTYWAĆ SHELL DO PRACY

- A może dodać obiekt do bazy danych:

```
>>> Pacjent.objects.create(Imię="Kasia", Nazwisko="Jakas")
<Pacjent: Kasia Jakas>
>>>
```

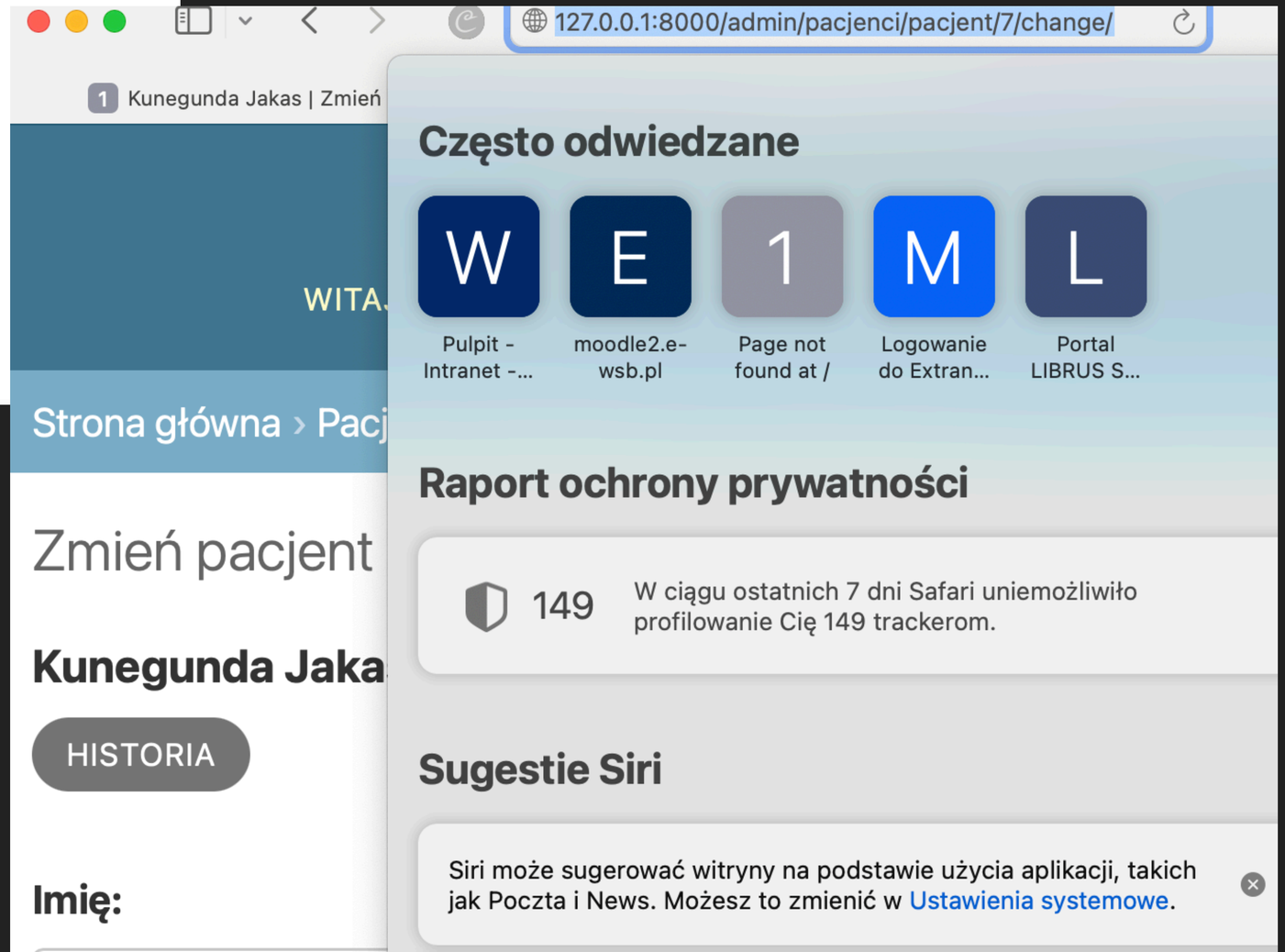
IMIĘ	NAZWISKO	DATA
Kasia	Jakas	-
UW	PU	23 listopada 2023
dd	dd	9 sierpnia 2023
Staszek	Kowalski	20 listopada 2023
Ola	Żarnecka	10 października 1981
Jan	Kowalski	10 października 1981
Łukasz	Piątek	16 listopada 2023

7 pacjents

WARTO WYKORZYSTYWAĆ SHELL DO PRACY – TO CAŁY CZAS ORM

- W shellu można też dość szybko wykonywać aktualizacje. Zamiast Save - delete

```
>>> dwa = Pacjent.objects.get(id=7)
>>> dwa
<Pacjent: Kasia Jakas>
>>> dwa.Imię ="Kunegunda"
>>> dwa.save()
>>>
```



WSZYSTKO CO ROBILIŚMY W SHELLU, MOŻNA STOSOWAĆ W METODACH

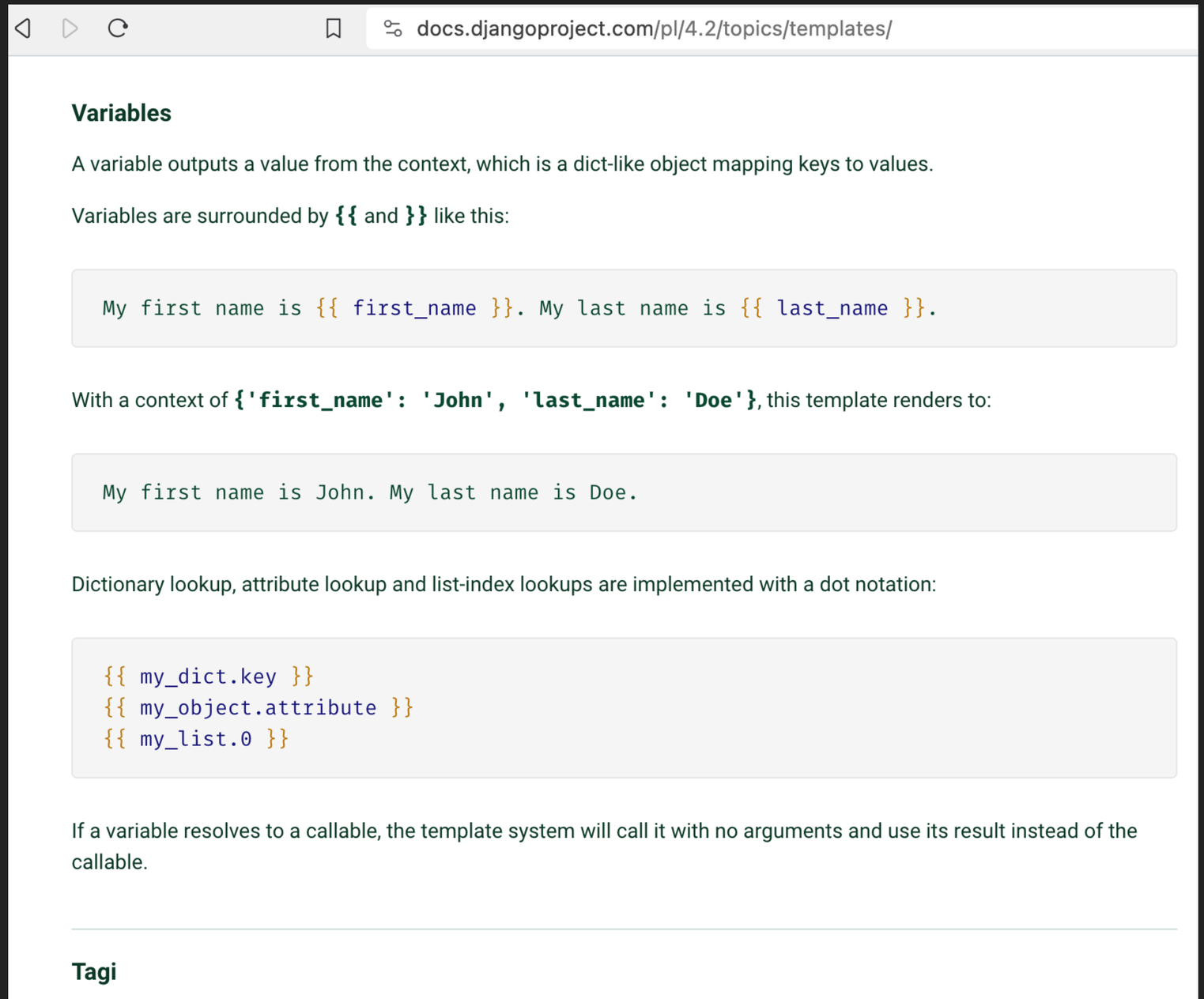
```
def spis_pacjentow(request):
    wszyscy = Pacjent.objects.all()
    return render(request, template_name: 'pacjenci.html', con
```

WRACAMY DO SZABLONÓW

!!!!

```
<h2>Znajdować się tutaj będzie baza pacjentów</
{{  }}


{% for imiona_pacjentów in imiona_pacjentów%}
<p>{{ imiona_pacjentów }}</p>
{% endfor %}
#dodane dynamiczne przekazywanie pojedynczych d
</body>
</html>
```

WRACAMY DO SZABLONÓW – WAŻNA DOKUMENTACJA – SĄ WSZYSTKIE SKŁADNIKI



WRACAMY DO SZABLONÓW – WAŻNA DOKUMENTACJA – SĄ WSZYSTKIE SKŁADNIKI

- W samym szablonie może jeszcze stosować wiele dodatków i filtrów. Przykład:

```
<body>
<h2>Ilość pacjentów w klinice: {{ imiona_pacjentów|length }}</h2>
<p><b>ten krok mi nie zadziałał...</b>b></p>
```

```
12      {% if imiona_pacjentów|length == 7 %}
13      <h2> Działa jak spełniony</h2>
14      {% endif %}
15      <h2>Ilość pacjentów w klinice: {{ imiona_pacjentów|length }}</h2>
16
```

WRACAMY DO SZABLONÓW – WAŻNA DOKUMENTACJA – SĄ WSZYSTKIE SKŁADNIKI

- Połączenie HTMLa i kodu Python dane coraz to ciekawsze efekty

```
<h2>Ilość pacjentów w klinice: {{ imiona_pacjentów|length }}</h2>
{% if imiona_pacjentów|length == 0 %}
<h2> Działa jak spełniony</h2>
{% else %}
    {% for imiona_pacjentów in imiona_pacjentów%}
        <p>{{ imiona_pacjentów }}</p>
    {% endfor %}
{% endif %}
```

WRACAMY DO SZABLONÓW – WAŻNA DOKUMENTACJA – SĄ WSZYSTKIE SKŁADNIKI

- Połączenie HTMLa i kodu Python dane coraz to ciekawsze efekty

```
def spis_pacjentow(request):
    wszyscy = Pacjent.objects.all() #można dodać tu count
    #ilosc =len(wszyscy)
    #wszyscy=[]   - tu jesy pusta tablica dla warunku
    return render(request, template_name: 'pacjenci.html', context: {'imiona_pacjentów':
```

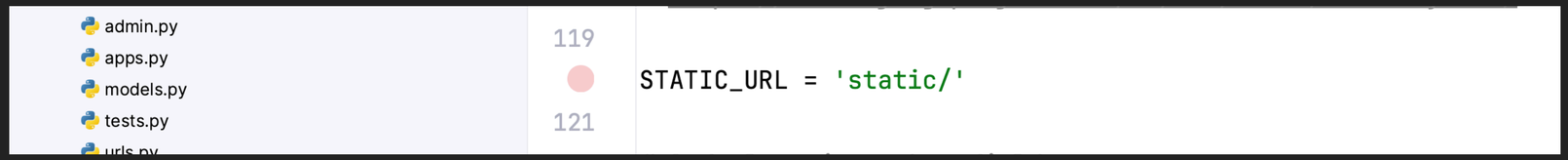
Ilość pacjentów w klinice: 0

Działa jak spełniony

DOŁĄCZANIE DO PROJEKTU PLIKÓW STATYCZNYCH

Pliki statyczne w Django to pliki, które są niezbędne do wyświetlenia pełnej strony internetowej, takie jak grafiki, JavaScript lub CSS. Pliki te nie są generowane dynamicznie przez serwer, ale są przechowywane w określonych katalogach i serwowane bez zmian. Django ma wbudowany mechanizm do zarządzania plikami statycznymi

DOŁĄCZANIE DO PROJEKTU PLIKÓW STATYCZNYCH - METODA PIERWSZA



W pierwszej metodzie upewniamy się, że katalog static jest dodany do ustawień.

Domyślnie jest URL, dobrze jeszcze dodać katalog z plikami. Sam katalog ręcznie należy później dodać

```
20  STATIC_URL = 'static/'
21  STATICFILES_DIRS = ['statyczne']
22  # Default primary key field type
```

DOŁĄCZANIE DO PROJEKTU PLIKÓW STATYCZNYCH - METODA PIERWSZA

Następnie tworzymy sobie jakiś plik statyczny, który załadujemy do szablonu.

Standardowo zaczyna się od css ;-) Sam plik style css działa jak klasyczny plik

CSS i nie trzeba nic kombinować. Ale uwaga. Podłączenie jest nieco bardziej zakręcone niż w przypadku samego i zwykłego html

```
1  {% load static %}
2  <!doctype html>
3  <html lang="pl">
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport"
7            content="width=device-width, user-scalable=no, initial-sca
8      <meta http-equiv="X-UA-Compatible" content="ie=edge">
9      <title>Document</title>
10     <link rel="stylesheet" href="{% static 'style.css' %}"
```

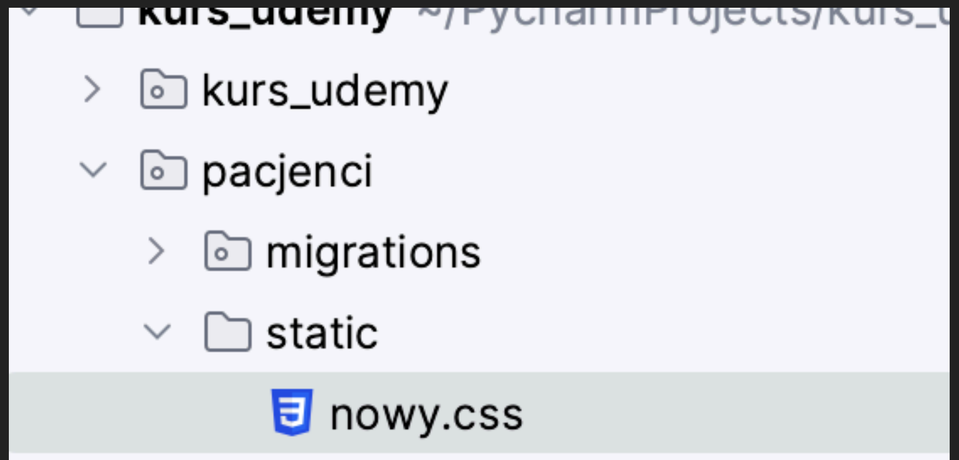
METODA PIERWSZA POLEGA NA TYM, ŻE TO STATIC JEST GLOBALNE I MOŻNA JE DZIELIĆ POMIĘDZY POSZCZEGÓLNYMI APLIKACJAMI W CAŁYM PROJEKCIE

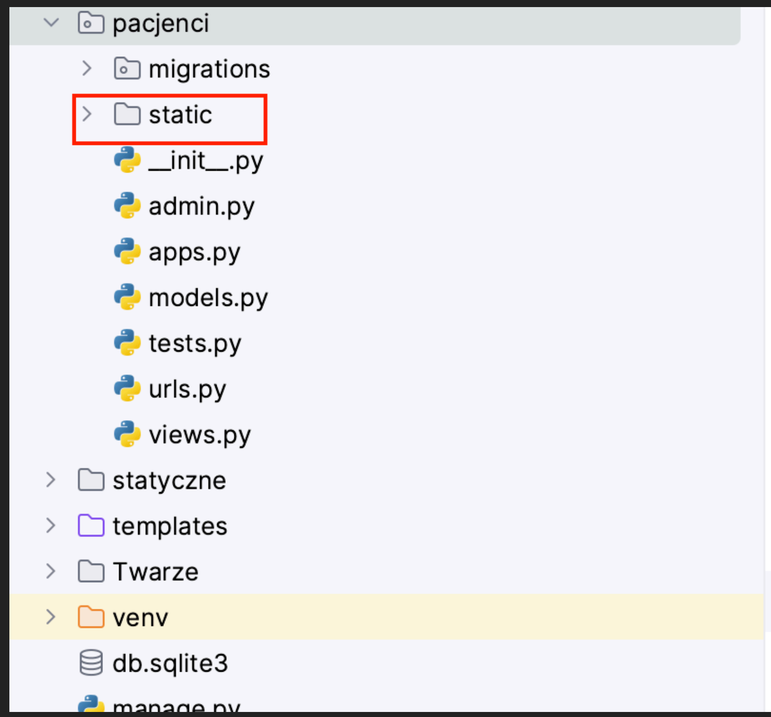
Metoda pierwsza

DOŁĄCZANIE DO PROJEKTU PLIKÓW STATYCZNYCH - METODA DRUGA

W drugiej metodzie możemy stworzyć katalogi static dla każdej aplikacji w projekcie

z osobna.






```
3   <html lang="pl">
4   <head>
5       <meta charset="UTF-8">
6       <meta name="viewport"
7             content="width=device-width, user-scalable=no, initial-sc
8       <meta http-equiv="X-UA-Compatible" content="ie=edge">
9       <title>Document</title>
10      <link rel="stylesheet" href="{% static 'nowy.css' %}"
11  </head>
12  <body>
13  <h2>Ilość pacjentów w klinice: {{ imiona_pacjentów|length }}</h2>
```

ZAMIAST PLIKÓW STATYCZNYCH MOŻNA PODŁĄCZAĆ PLIKI MULTIMEDIALNE

Tu też w settings w razie W trzeba dodać ścieżkę do katalogu z mediami

Oryginalnie:

MEDIA_URL = ‚media‘

Oraz MEDIA_ROOT - podstawowe miejsce na przechowywanie plików

Folder, który podamy w ROOT, trzeba oczywiście stworzyć w głównej strukturze projektu

Wcześniej w przykładach modeli było pole graficzne - tam robiłem upload to ‚twarze‘ - i mi się zrobił katalog twarze. Teraz jak jest zdefiniowany katalog na media, to tam się to będzie robiło - coś na kształt porządku - później jeszcze trzeba popracować nad ich wyświetlaniem, bo sam serwer Django tego nie lubi. NIE ZALECA SIĘ UŻYWAĆ DJANGO DO WYŚWIETLANIA PLIKÓW

ZAMIAST PLIKÓW STATYCZNYCH MOŻNA PODŁĄCZAĆ PLIKI MULTIMEDIALNE

```
2   from django.contrib import admin
3   from django.urls import path, include
4   from django.conf import settings
5   from django.conf.urls.static import static
6   #from pacjenci.views import test_response
7   urlpatterns = [
8       path('admin/', admin.site.urls),
9       path('pacjenci/', include('pacjenci.urls')) #to prowadzi do urls w aplikacji
10      ]+ static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
11
```

NA SERWER ZALECA SIĘ ROZWIĄZANIA ZEWNĘTRZNE

Do serwowania plików